KURS PROGRAMOWANIA

W tym kursie będziemy stopniowo poznawać tajniki programowania. Każdy odcinek to inny projekt, opisany krok po kroku, który wprowadza początkującego programistę w znane środowiska programistyczne

Programowanie – choć jest tylko jednym słowem, kryje się pod nim ogrom znaczeń, a raczej sposobów na to, jak jest ono postrzegane. W tym kursie zobaczymy różne oblicza programowania. Każdy z odcinków będzie przedstawiał inne środowisko programistyczne i język programowania. Po to, by nauczyć się tego, co w programowaniu uniwersalne i wspólne dla różnych środowisk i języków. Zaczniemy od zagadnień prostszych, aby wykonywać kolejne kroki w głąb programistycznego świata. Będziemy płynnie przechodzić od środowisk edukacyjnych do profesjonalnych języków (między innymi C#, Visual Basic) i narzędzi (między innymi Visual Studio, Unity Engine). Środowiska edukacyjne są oczywiście prostsze do opanowania od profesjonalnych – jednak w pełni funkcjonalne. Pozwalają na tworzenie gier i aplikacji dokładnie takich, jakie sobie wymyślimy.

kj

TEMATY KURSU

1. **App Inventor** Jak tworzyć gry mobilne
2. **Small Basic** Jak zrobić Tetris
3. **Visual Basic** Wstęp do profesjonalnych języków programowania
4. **Visual Basic** Platforma .NET
5. **C#** Język do wielu zastosowań
6. **Unity** Jak zrobić własną grę z grafiką 3D

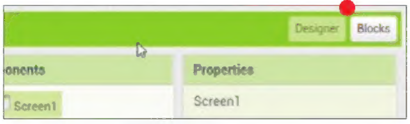
Fot. Andrey Suslov/istockphoto.com

APP INVENTOR: JAK TWORZYĆ GRY MOBILNE

Kurs rozpoczynamy od środowiska **App Inventor**. To oprogramowanie, dostępne z poziomu przeglądarki, pozwala na tworzenie gier i aplikacji przeznaczonych do instalacji na urządzeniach mobilnych z systemem Android (trwają beta testy wersji do aplikacji na iOS).
Jest coś, co wyróżnia App Inventora od pozostałych narzędzi i języków, które zostaną omówione w kolejnych częściach kursu. Programując w App Inventorze, nie musimy pisać skryptów – możemy układać je z puzzli. Czyni to App Inventora idealnym środowiskiem do rozpoczęcia przygody z programowaniem. Dzięki wykorzystaniu puzzli unikniemy literówek w skryptach, które są częstą bolączką początkujących programistów.

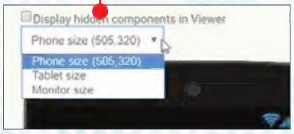
OKNO PROGRAMU

Zanim przejdziemy do tworzenia własnej gry, poznajmy podstawowe pojęcia związane z programem App Inventor – ułatwi to nam dalszą pracę.
Okno programu App Inventor składa się z dwóch głównych ekranów. Pierwszy z nich to **Designer**, gdzie modelujemy wygląd naszej aplikacji, a drugi to **Blocks**, gdzie przypisujemy jej działanie, tworząc skrypty zbudowane z puzzli zwanych tu blokami. Pomiędzy ekranami przełączamy się, klikając na przyciski odpowiadające ich nazwom, widoczne w prawym górnym rogu okna.

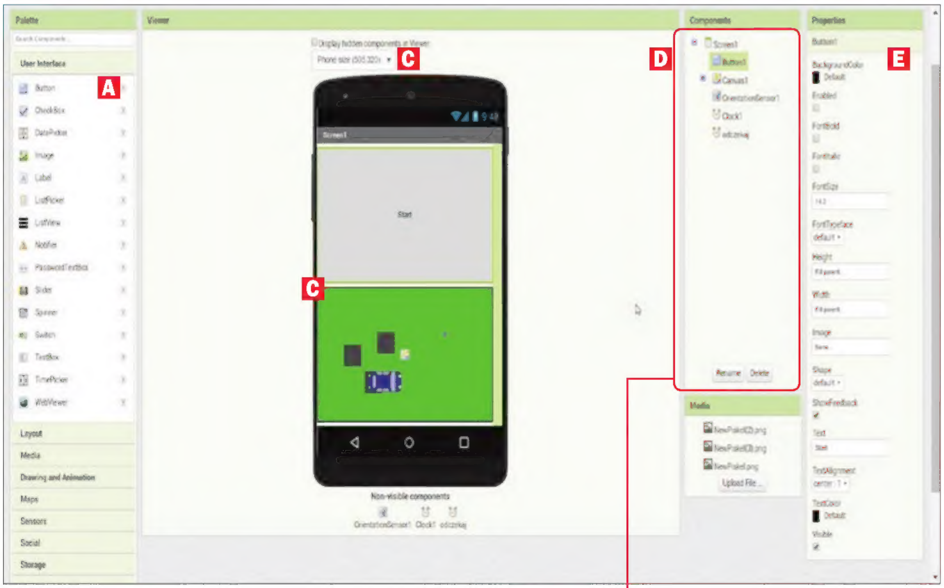


DESIGNER

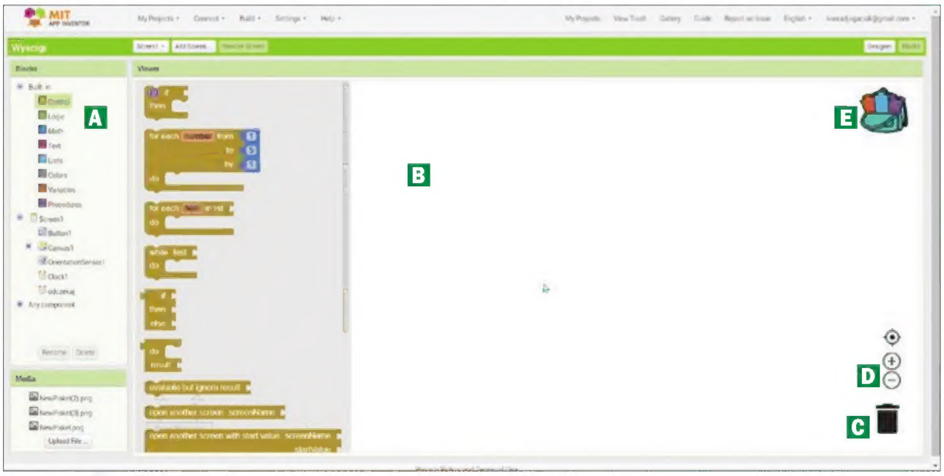
Zajmijmy się najpierw ekranem Designer. Pierwsza od lewej strony sekcja to paleta komponentów **A**. Komponenty to elementy, które umieszczamy w programie – na przykład przyciski, pola tekstowe – ale też moduły pozwalające na wykorzystanie map, użycie dyktafonu, aparatu czy żyroskopu.
Potrzebne nam komponenty przeciągamy na ekran podglądu **B** – on pokazuje nam, jak będzie wyglądała nasza aplikacja, kiedy włączymy ją na urządzeniu z Androidem.
Nieco ponad podglądem widzimy niewielką rozwijaną listę **C**. Pozwala ona na wybranie



docelowego urządzenia, na jakim ma być uruchamiana nasza aplikacja. Do wyboru mamy telefon, tablet i monitor – zmiana urządzenia powoduje jedynie zmianę wielkości ekranu podglądu i nie wpływa na efekt naszej pracy. Nawet jeśli pracujemy na podglądzie dla tabletów, stworzone przez nas aplikacje i tak będzie można uruchomić na telefonie.
Kolejna sekcja – po prawej stronie podglądu – to zestawienie dodanych do projektu komponentów **D**. Kiedy dodamy komponent do programu, wyświetli się on na liście. Wygląd i działanie komponentu możemy zmieniać poprzez zmianę wartości jego właściwości **E**. Są one wypisane w ostatniej z sekcji, skrajnie po prawej stronie.



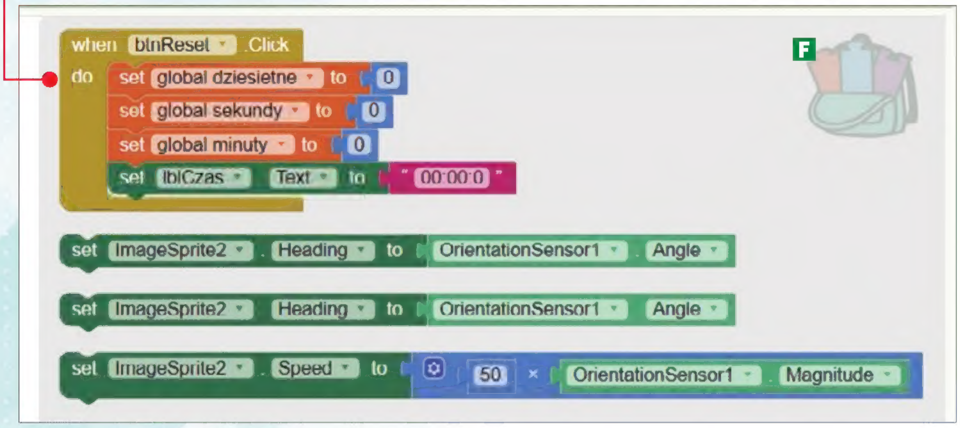
jest wyświetlany w programie tak samo jak standardowe kategorie dostępnych puzzli. W ten właśnie sposób komponenty poszerzają możliwości oprogramowania, dając nam do dyspozycji dodatkowe puzzle. W prawym dolnym rogu widzimy ikonę kosza na śmieci **C**.



BLOCKS

Po przejściu do sekcji Blocks po lewej stronie okna widzimy kategorie **A** dostępnych bloków i duże białe pole **B**. Wybierając kategorię, widzimy dostępne w niej puzzle, które możemy przeciągać na białe pole, tworząc w ten sposób skrypty opisujące działanie naszej aplikacji. Każdy z dodanych do projektu komponentów,

Przeciągając na nią umieszczone na białym polu skrypty, możemy je usunąć.
Powyżej kosza widoczne są przyciski z plusem i minusem **D**. Służą one do powiększania i pomniejszania puzzli.
Jeszcze nieco wyżej widzimy ikonę plecaka **E**. Plecak to mechanizm, dzięki któremu możemy przenosić skrypty pomiędzy projektami. Jeśli



Wyrwij, użyj dziurkacza i zbieraj do segregatora

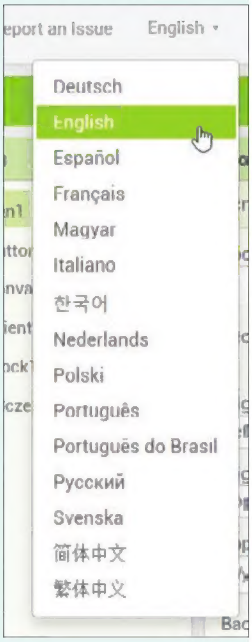
przeciągniemy skrypt do plecaka, zostanie on do niego dodany. Klikając na ikonę plecaka, otwieramy go, czyli wyświetlamy znajdujące się w nim skrypty **F**. Jeśli chcemy wykorzystać skrypt znajdujący się w plecaku w swoim projekcie, przeciągamy go z plecaka na obszar roboczy. Zawartość plecaka przypisana jest do konta Google, z którego logujemy się do programu.

Przed przystąpieniem do tworzenia gier dobrze jest zapoznać się z dostępnymi puzzlami (blokami), aby łatwiej było nam się poruszać po oprogramowaniu.

Po najechaniu na blok wyświetlona zostanie podpowiedź, opisująca jego działanie.

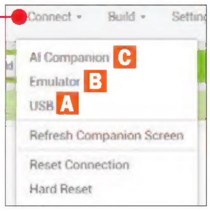
Wersja językowa

App Inventor ma polską wersję językową. Nie jest ona jednak polecana dla osób, które chcą wykorzystać to oprogramowanie jako jeden z pierwszych kroków do dalszej nauki programowania. Zdecydowana większość języków programowania jest oparta na języku angielskim. Bloki, z jakich korzystamy w App Inventorze do budowania skryptów, są odzwierciedleniem programistycznych instrukcji, które spotkamy potem w innych narzędziach, podczas pisania skryptów. Dzięki wykorzystaniu bloków w języku angielskim łatwiej zauważymy analogie pomiędzy blokami a odpowiadającymi im programistycznymi instrukcjami.



JAK TESTOWAĆ GRĘ

Podczas tworzenia gier warto testować poszczególne funkcje wprowadzane przez nas do projektu. Aby móc to zrobić, niezbędne jest nawiązanie połączenia pomiędzy App Inventorem a fizycznym urządzeniem z Androidem bądź emulatorem Androida. Opcje służące do połączenia uruchomimy po wybraniu z menu pozycji **Connect**.



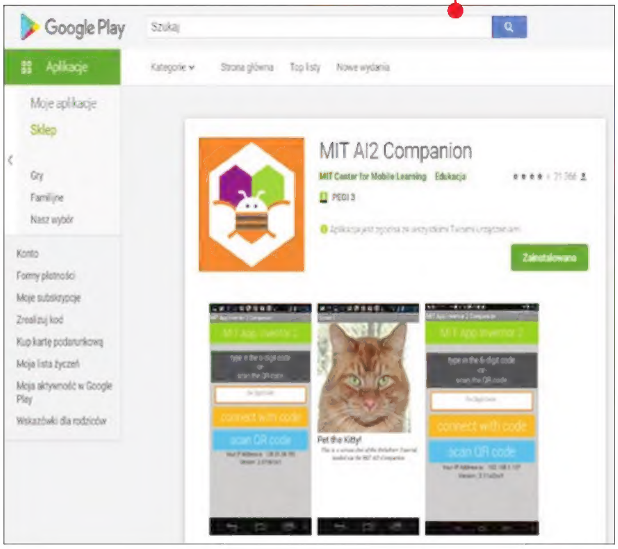
- Opcja **USB** **A** pozwala na przetestowanie tworzonego przez nas projektu na podłączonym urządzeniu z Androidem, w którym włączono tryb debuggowania.
- Opcja **Emulator** **B** pozwala na przetestowanie projektu za pomocą wirtualnego urządzenia. Niezbędna jest do tego instalacja odpowiedniego emulatora – **aiStarter**.
- Rekomendowanym przez twórców App Inventora rozwiązaniem jest jednak wykorzystanie opcji **AI Companion** **C**.

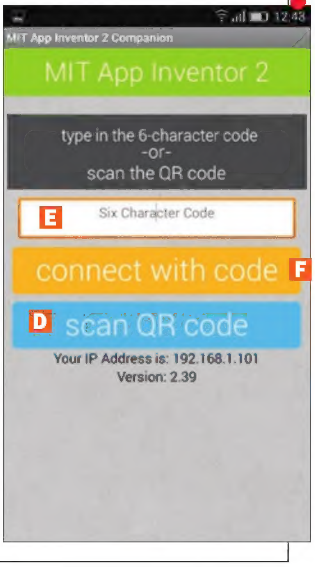
1 Po jej wybraniu wyświetlony zostanie kod QR.

2 Możemy zeskanować go poprzez aplikację **MIT AI2 Companion** (do pobrania ze Sklepu Play).

3 Skaner uruchamiamy za pomocą przycisku **Scan QR code** **D**. Możemy również ręcznie wpisać tekst wyświetlony obok kodu QR w pole **Six Character Code** **E**.

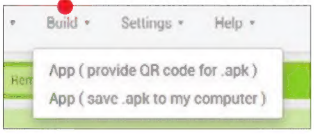
4 Następnie, naciskając **Connect with code** **F**, łączymy się z naszym projektem. Co ważne – wprowadzane przez nas w projekcie zmiany będą od razu widoczne w testowanej aplikacji.



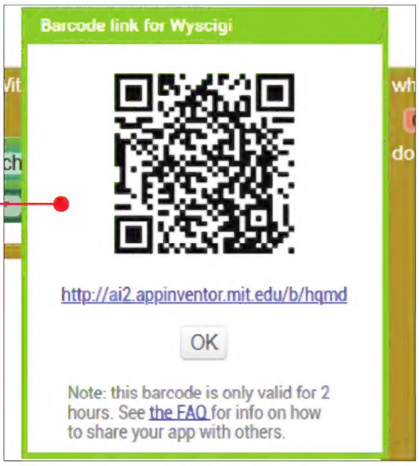


JAK ZAINSTALOWAĆ GOTOWĄ GRĘ NA URZĄDZENIU Z ANDROIDEM

Aby zainstalować aplikację na urządzeniu z systemem Android, niezbędne jest posiadanie pliku **APK**. Do jego wytworzenia możemy wykorzystać opcje, które znajdziemy w menu **Build**.



1 Pierwsza z nich – **App (provide QR code for .apk)** – pozwala na wygenerowanie kodu QR, który podobnie jak w przypadku testowania, możemy zeskanować przez aplikację MIT AI2 Companion. Pozwala to na pobranie



pliku APK do naszego urządzenia mobilnego i rozpoczęcie instalacji. W przypadku plików APK możliwe jest skanowanie kodu QR także innymi skanerami – ponieważ kod prowadzi po prostu do pliku z projektem, więc niezależnie od użytego skanera będziemy mogli pobrać ten plik.

Kod QR jest aktywny przez dwie godziny od momentu wygenerowania.

2 Druga z widocznych w menu opcji – **App (save .apk to my computer)** – generuje plik APK i pozwala na zapisanie go na dysku naszego komputera. Taki plik przesyłamy w dowolny sposób do urządzeń mobilnych i otwierając go, uruchamiamy instalację.

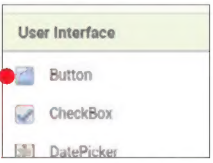
WYŚCIGI SAMOCHODOWE NA ANDROIDA

Kiedy wiemy już, do czego służą poszczególne elementy programu, możemy przejść do tworzenia naszej pierwszej gry mobilnej. Będą to **Wyścigi samochodowe**. Zadaniem gracza będzie dojechanie do mety w jak najkrótszym czasie, uważając na to, by nie wyjechać poza trasę. Samochód gracza będzie sterowany poprzez odpowiednie nachylenie urządzenia.

App Inventor 2 pozwala na tworzenie skryptów w projekcie poprzez interfejs oparty na puzzlach, które podzielone są na kategorie. Niektóre z puzzli są jednak przypisane do poszczególnych komponentów. Wykorzystanie takich puzzli jest możliwe dopiero po przeciągnięciu komponentów do projektu. Dlatego najpierw należy przygotować wygląd okna naszej gry, a dopiero potem przejść do tworzenia skryptów. Tego rodzaju podejście występuje także w wielu profesjonalnych środowiskach programistycznych.

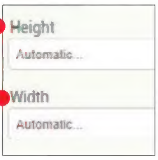
MODELOWANIE OKNA GRY

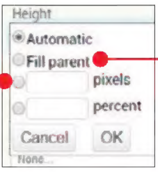
Zadajmy sobie pytanie: Co ma zobaczyć użytkownik naszej gry, tuż po jej włączeniu? Dobrze, aby był to przycisk, którym można rozpocząć rozgrywkę.



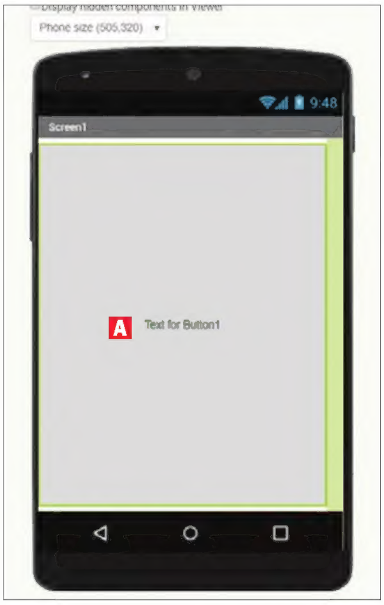
1 By go stworzyć, wykorzystamy tradycyjny przycisk – komponent **Button** ●. Znajdziemy go w palecie komponentów, w zakładce **User Interface**. Przeciągamy przycisk na podgląd okna programu ●.

2 Przycisk nie przypomina jeszcze tego, co chcemy uzyskać w naszej grze. Chcielibyśmy, aby przycisk miał inną formę. By był możliwie dużym prostokątem, który wypełnia cały ekran. Po zaznaczeniu odpowiedniego komponentu na oknie podglądu projektu, możemy modyfikować jego właściwości (**Properties**). Tak też musimy zrobić z naszym przyciskiem. Pierwsza zmiana to wymiary przycisku. Pola **Height** i **Width** ● pozwalają nam na określenie odpowiednio wysokości i szerokości przycisku. Możemy na przykład wpisać tam konkretne wymiary, wyrażone w pikselach ●. Niestety, nie znamy wymiarów ani rozdzielczości ekranów, na jakich uruchamiana będzie nasza gra. Możemy jednak sprawić, by gra wyglądała dobrze na każdym urządzeniu, na jakim zostanie uruchomiona, by przycisk wypełniał cały ekran, dopasowując swoją wielkość do jego rozmiarów. Zarówno wysokość, jak i szerokość przycisku powinna dostać wartość **Fill parent** ●. Przycisk wypełnia teraz cały obszar ekranu gry **A**.

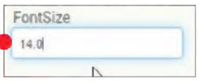




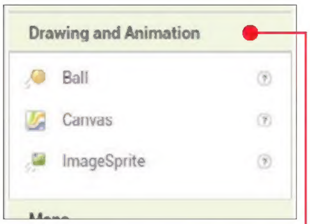
3 Kolejną zmianą, jaką należy wprowadzić, jest usunięcie z przycisku napisu **Text for Button1** i zastąpienie go pasującym do gry słowem – na przykład **Start** lub **Rozpocznij**. W tym celu usuwamy we właściwościach przycisku tekst wpisany w pole **Text**. Zastępujemy go napisem, jaki chcemy widzieć na przycisku.

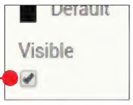


4 Wyświetlony na przycisku tekst jest dość mały. Możemy powiększyć go, wpisując interesujący nas rozmiar czcionki w pole **FontSize** ● we właściwościach przycisku.



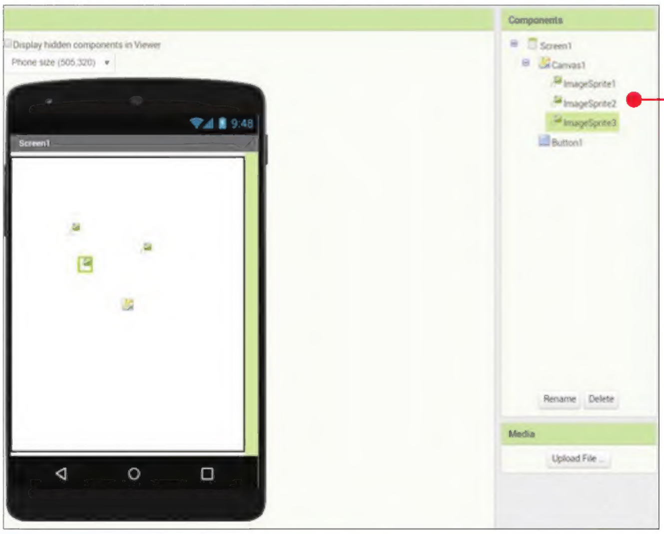
5 W ten sposób przycisk do rozpoczynania naszej gry jest już gotowy. Co dalej powinno znaleźć się na ekranie? Przycisk powinien zniknąć, a powinna go zastąpić trasa, którą musimy przejechać, nasz samochód i meta – innymi słowy, główne elementy graficzne naszej gry. Takie elementy gry nazywamy „duszkami” (ang. sprite). Duszki to komponenty o nazwie **ImageSprite**. Jeśli spróbujemy przeciągnąć komponent znajdujący się w kategorii **Drawing and Animation** ● bezpośrednio do okna, w którym modelujemy nasz projekt, nie zostanie on dodany. Komponenty **ImageSprite** można dodawać tylko wewnątrz komponentu **Canvas**. Dlatego to właśnie Canvas należy dodać najpierw do naszego projektu. Zanim dodamy Canvas, trzeba byłoby jeszcze ukryć przycisk. Możemy to zrobić, usuwając w jego właściwościach zaznaczenie przy pozycji **Visible** ●.





6 Gdy przycisku już nie widać, można dodać Canvas do projektu. Aby gra była widoczna na całym ekranie urządzenia mobilnego, Canvas powinien mieć wymiary takie jak przycisk – wypełniając cały możliwy obszar. Dlatego we właściwościach obiektu **Canvas** zmieniamy jego wysokość i szerokość (odpowiednio Height i Width) na **Fill parent**, jak w przypadku przycisku.

7 Na powiększony do pełnych rozmiarów **Canvas** przeciągamy trzykrotnie komponent **ImageSprite** ●. Te duszki wykorzystamy do budowy toru, po którym będzie jeździł nasz samochód. Każdy z tych trzech duszków będzie fragmentem toru, a poprzez skrypty ustalimy jego dokładny wygląd. Na

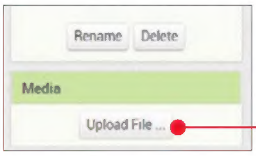


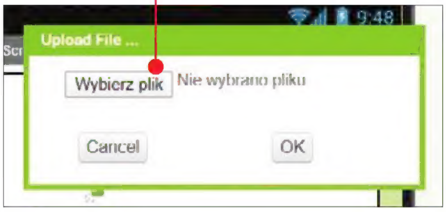
Piskel

Do tworzenia grafik polecamy oprogramowanie Piskel (**W KŚ+**). Jest to edytor graficzny stworzony z myślą o pixel art (sposób tworzenia grafiki rastrowej za pomocą programów pozwalających na edytowanie obrazów na poziomie pojedynczych pikseli).

tym etapie niezbędne będzie jednak wczytanie grafiki do tych obiektów. Najlepiej przygotować plik graficzny jednolitego koloru – takiego, jakiego chcemy mieć trasę.

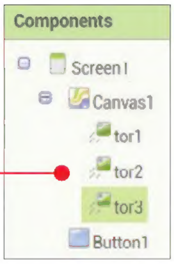
8 Plik graficzny należy wczytać do App Inventora. Robimy to w niewielkiej sekcji **Media**, która znajduje się poniżej listy dodanych już do projektu komponentów. Należy kliknąć na przycisk **Upload file**, by otworzyć okno importu plików, w którym klikamy na **Wybierz plik**, co otwiera okno dialogowe, z poziomu którego możemy wczytać plik graficzny.



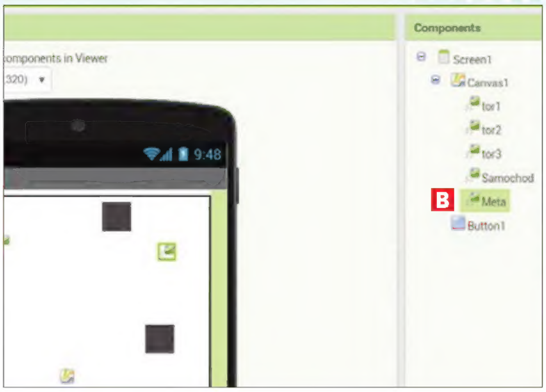


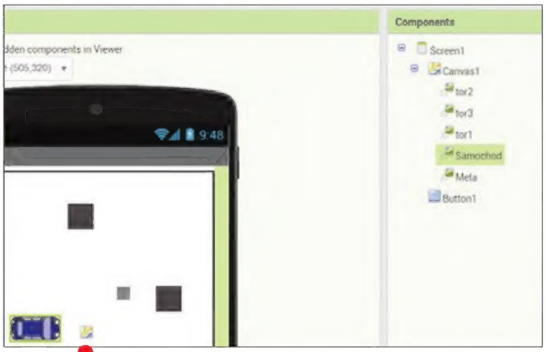
9 Gdy plik jest wczytany i zatwierdzony, możemy przypisać go do duszków. W każdym z nich powinniśmy we właściwości **Picture** wybrać wczytany plik graficzny.

10 Pod listą dodanych komponentów, a nad sekcją **Media**, znajduje się przycisk **Rename**. Jego użycie pozwala na zmianę nazwy zaznaczonego komponentu dodanego do projektu. Zmieniamy nazwy duszków, z których będziemy budować tor, na tor1, tor2 i tor3.



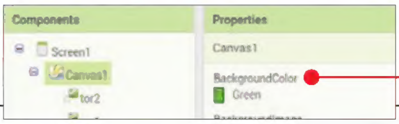
11 Dodajemy dwa kolejne komponenty typu **ImageSprite** i zmieniamy im nazwy na **Samochod** i **Meta** **B**.





12 Wczytujemy dwa kolejne pliki graficzne – widziany z lotu ptaka samochód (zwrócony w prawą stronę) i metę. Meta może być kwadratem w biało-czarną szachownicę. Wczytanie grafiki przypisujemy (poprzez właściwość **Picture**) do dodanych w poprzednim kroku duszków.

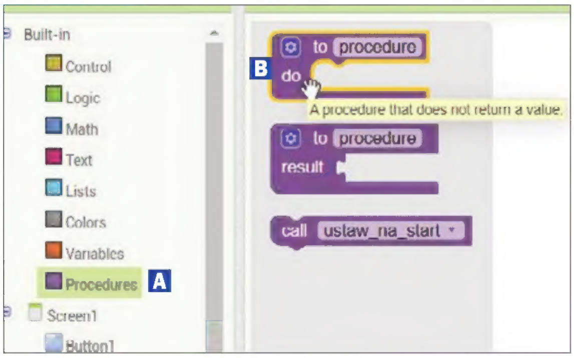
13 Dla obiektu **Canvas** możemy jeszcze zmienić wartość właściwości **BackgroundColor** – mówiącej o kolorze tła. Wybieramy kolor zielony, aby tło symbolizowało trawę wokół drogi.



UKŁADANIE SKRYPTÓW

UŁOŻENIE OBIEKTÓW W OKNIE

Przechodzimy do ekranu **Blocks** – to w nim należy ułożyć skrypty opisujące działanie gry.



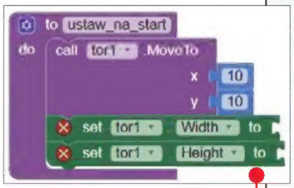
1 Zaczniemy od odpowiedniego modelowania toru. Z trzech kwadratów musimy stworzyć tor do jazdy. Tor powinien mieć kształt odwróconej litery U – z dwoma pionowymi odcinkami po bokach i jednym, poziomym, u góry okna.

2 Ułożymy procedurę, która zmieni rozmiar obiektów tor1, tor2 i tor3, rozciągając je, i przesunie je w odpowiednie miejsca, tak by utworzyły opisany wcześniej tor. By stworzyć procedurę z kategorii **Procedures** **A**, bierzemy blok **to procedure do** **B** i przeciągamy go na pusty obszar roboczy.

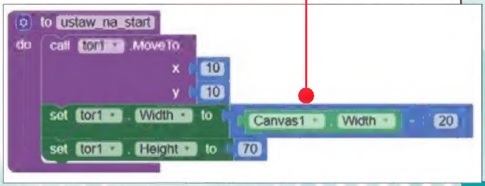
3 W edytowalnym polu tekstowym na bloku wpisujemy nazwę procedury: **ustaw_na_start**.

4 Do wnętrza procedury wkładamy blok **call tor1.MoveTo**, a do niego przyłączamy podwójnie (w pola **x** i **y**) blok pozwalający na wpisanie liczby, którą znajdziemy w kategorii **Math** (gdzie domyślnie wpisane jest 0). Pole **x** mówi o odległości duszka od lewej krawędzi okna, pole **y** zaś o odległości duszka od górnej krawędzi okna. W obydwa pola wpisujemy liczbę **10**. Pierwszy element toru będzie poziomą linią u góry okna.

5 Pierwszy fragment toru jest już przesunięty w odpowiednie miejsce, jednak trzeba mu jeszcze nadać odpowiednie wymiary. Zrobimy to, używając **set tor1.Width to** i **set set tor1. Height to**, które służą odpowiednio do ustawiania szerokości oraz wysokości duszka.

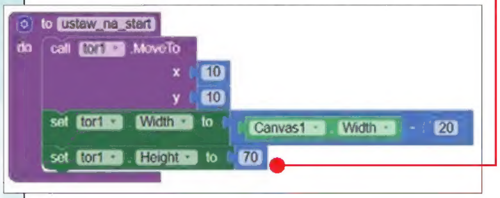


6 Ponieważ ustawiliśmy nasz tor 10 pikseli od lewej krawędzi, aby był on symetryczny, powinien się kończyć 10 pikseli przed prawą krawędzią. A dwa razy po 10 pikseli po bokach daje nam 20 pikseli. Szerokość duszka powinna być 20 pikseli mniejsza niż szerokość obiektu **Canvas**. Możemy to ułożyć za pomocą bloku odejmowania, który znajduje się w kategorii **Math**, a także bloku **Canvas1.Width** (zawierającego szerokość

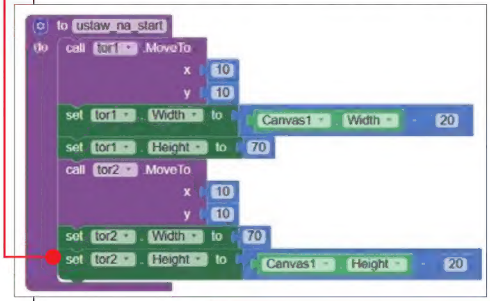


kanwy), znajdującego się w kategorii **Canvas1** i bloku z liczbą do wpisania.

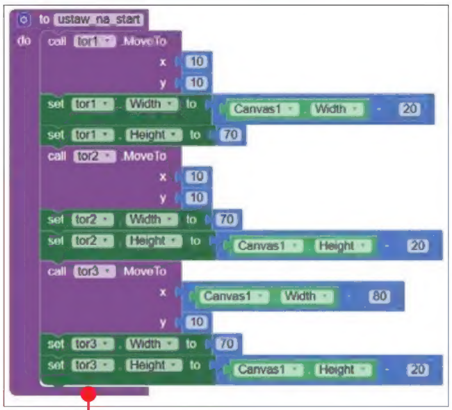
7 Również wysokość obiektu **tor1** powinniśmy ustalić. Tu możemy zdecydować się na wpisanie odpowiadającej nam liczby – **70** pikseli powinno być odpowiednie.



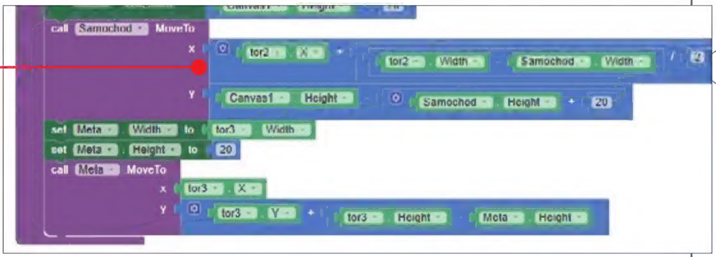
8 Analogicznie do poziomej części toru, w obiekcie **tor2** powinniśmy ustawić lokalizację i wymiary tak, by ten stał się pionową częścią toru po lewej stronie okna gry.



9 Także trzeci fragment toru powinniśmy ustawić po prawej stronie, w takim miejscu, by utworzył on symetryczny obraz do linii po lewej stronie.



10 Również **Samochod** i **Meta** powinny być przeniesione w odpowiednie miejsca. Samochód powinien znaleźć się na dole prostej po lewej stronie. Obliczamy jego **x** i **y**, tak by znalazł się on idealnie na środku, jeśli chodzi o szerokość, i by grafika samochodu kończyła się 10 pikseli przed końcem toru, jeśli chodzi o pionową oś. Obliczenia należy ułożyć z bloków znajdujących się w kategorii **Math**.

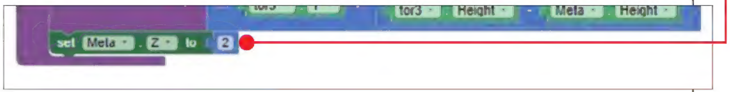


11 Meta powinna znaleźć się na drugim końcu toru – niech ma ona 20 pikseli wysokości i szerokość taką jak tor. Powinna być ustawiona w takim miejscu, by jej dolna krawędź pokrywała się z dolną krawędzią toru.

Pamiętaj

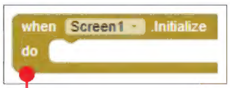
Jeśli masz grafikę samochodu w rozmiarze innym niż 60 na 30 pikseli – zmień wymiary obiektu **Samochod** poprzez użycie odpowiednich bloków – tak jak zmieniasz wymiary torów – niech ma on 60 pikseli szerokości i 30 pikseli wysokości.

12 Dla mety należy ustawić też właściwość **Z** – poprzez blok **set Meta.Z to** – na wartość **2**. Domyślnie każdy **ImageSprite** ma właściwość **Z** ustawioną na **1**. Im większa jest tam wartość, tym „wyżej" jest obiekt. Jeśli meta ma zasłaniać końcówkę toru, powinna mieć **Z** większe niż on. Inaczej meta będzie przysłonięta torem.



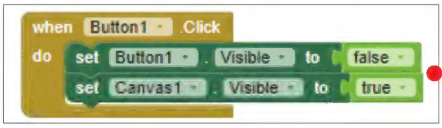
UKRYCIE CANVAS PRZED STARTEM GRY

Kolejnym krokiem w tworzeniu naszej gry będzie sprawienie, by na początku, tuż po uruchomieniu, widoczny był jedynie przycisk, a **Canvas** był ukryty.



1 W tym celu korzystamy z bloku **when Screen1.Initialize**, który zgłasza zdarzenie uruchomienia programu. Bloki o tym kolorze, których nie da się od dołu i od góry przyczepić do innych bloków, to zdarzenia. Zdarzenia mówią o wystąpieniu czegoś w programie i pozwalają nam na reakcję na tego czegoś wystąpienie. To co włożymy do bloku **when Screen1.Initialize**, wykona się, gdy uruchomimy grę.

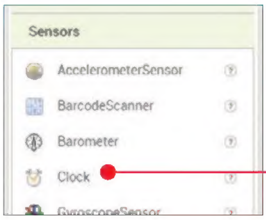
2 Właściwość **Visible** zarówno dla **Button1**, jak i **Canvas1** może przyjąć tylko jedną z dwóch wartości – true lub false. Bloki z tymi wartościami znajdują się w kategorii **Logic**. Pierwszy z nich oznacza prawdę, a drugi fałsz. Blok **false** przyczepiamy do **set Button1.Visible to**, a blok **true** doczepiamy do ustawienia widoczności obiektu **Canvas1**.



3 Przycisk ma zniknąć, gdy na niego klikniemy, a **Canvas** ma się wtedy pojawić. Z kategorii **Button1** bierzemy blok **when Button1.Click do** i układamy w nim skrypt ukrywający przycisk i pokazujący obiekt **Canvas1**.

DODAWANIE NIEWIDOCZNYCH KOMPONENTÓW

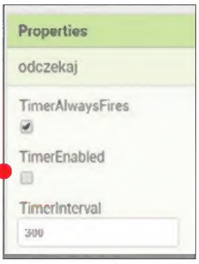
Gdybyśmy zaraz pod blokiem odpowiedzialnym za pokazywanie komponentu **Canvas** uruchomili procedurę **ustaw_na_start**, obiekty mogłyby pojawić się w nieodpowiednich miejscach. To przez opóźnienia programu, jeśli w skrypcie odczytujemy z komponentu **Canvas** wymiary, mogą być one jeszcze niezaktualizowane w momencie wywoływania procedury i mogą zostać źle odczytane. Powinniśmy zatem odczekać chwilę i dopiero po pewnym czasie uruchomić procedurę. App Inventor nie ma bloku **poczekaj**, musimy ten mechanizm stworzyć sami. W tym celu powracamy do ekranu **Designer**.



1 Niezbędne okaże się dodanie jeszcze jednego komponentu – **Clock** z kategorii **Sensors**.

2 Po przeciągnięciu go na okno podglądu gry nie zostanie on wyświetlony w oknie,

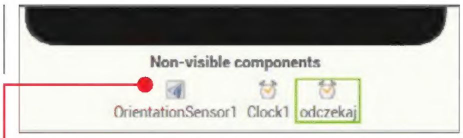
ale pod nim, jako niewidoczny komponent. Mimo że jest niewidoczny, daje nam do dyspozycji nowy zestaw bloków do wykorzystania w układaniu skryptów. Jednak zanim przejdziemy do bloków, możemy zmienić nazwę tego komponentu na zgodną z jego przeznaczeniem – czyli „odczekaj”. Należy też nieco zmienić jego właściwości.

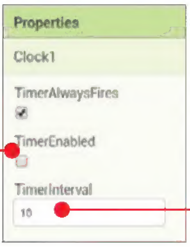


3 Właściwość **TimerEnabled** nie powinna być zaznaczona, by komponent odliczał czas nie od startu programu, ale od włączenia go z poziomu skryptów.

4 Właściwość **TimerInterval** powinna mieć wartość 300, by komponent odliczył 0,03 sekundy, a nie całą sekundę (wpisujemy czas wyrażony w milisekundach).

Jeśli jesteśmy już przy dodawaniu niewidocznych komponentów, potrzebne będą jeszcze dwa. Komponent **Clock** wykorzystamy jako „pętlę” do ciągłego sprawdzania, czy samochód znajduje się na torze jazdy, a **OrientationSensor** (również w kategorii **Sensors**) do sterowania samochodem poprzez pochylanie urządzenia.





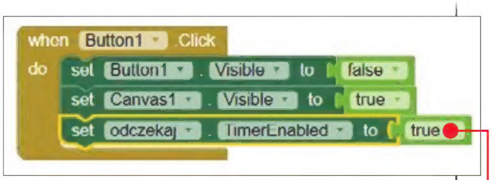
1 W kolejnym Clocku także właściwość **TimerEnabled** nie powinna być zaznaczona, by komponent odliczał czas nie od startu programu, ale od włączenia go z poziomu skryptów.

2 Właściwość **TimerInterval** powinna mieć wartość **10**, by komponent mógł wykonywać „pętlę” możliwie często.

3 Właściwości komponentu **OrientationSensor** z poziomu okna **Designer** powinny pozostać niezmienione.

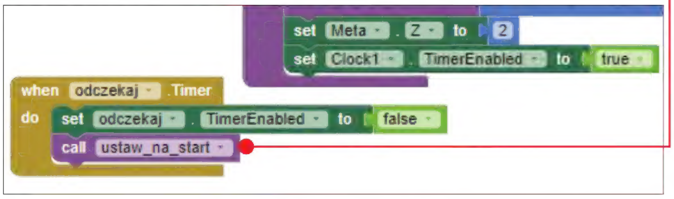
Wracamy do bloków.

1 Teraz musimy zająć się uruchomieniem „odczekania” – blok **set odczekaj.TimerEnabled to** powinien wykonywać się w momencie naciśnięcia przycisku, i włączać ten



komponent, czyli powinna być przyczepiona do niego wartość true.

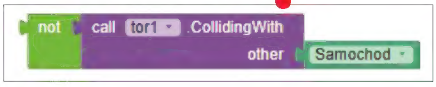
2 Z kategorii **odczekaj** wkładamy blok **when odczekaj.Timer do**. To co do niego włożymy, wykona się 0,03 s po wciśnięciu przycisku – wtedy powinna zostać uruchomiona ułożona wcześniej procedura. Wkładamy do tego bloku blok **call ustaw_na_start**. Oprócz tego wyłączamy ten Clock, by nie wykonywał on procedury ciągle, ale tylko raz.



3 Dodatkowo na końcu procedury **ustaw_na_Start** dodajemy instrukcję, która uruchomi **Clock1**, czyli zegar, który co 0,01 s będzie wykonywał swoje zdarzenie **.Timer()**. Co w nim ułożymy – to za chwilę, najpierw zajmijmy się sterowaniem samochodem.

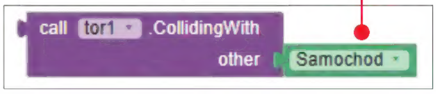
STEROWANIE SAMOCHODEM

Sterowanie samochodem zrobimy dzięki dodaniu komponentu **OrientationSensor**. Ma on w sobie blok zdarzenia **.OrintetionChanged** – przenosimy go na obszar roboczy. Wewnątrz niego należy ustawić kierunek (Heading) i prędkość (Speed) samochodu. Powinna ona być ustawiona na podstawie odczytów z **OrientationSensora** w sposób przedstawiony na rysunku.



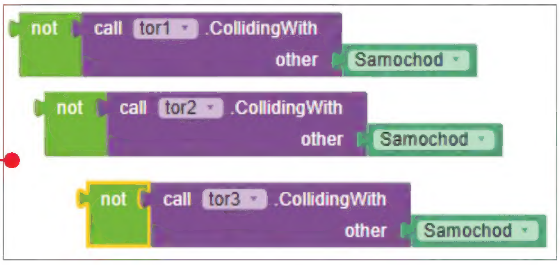
Samochód powinien wracać na start, jeśli nie dotyka toru. Ponieważ nasz tor podzielony jest na trzy obiekty – samochód nie dotyka „toru”, gdy nie dotyka obiektu **tor1**, i nie dotyka obiektu **tor2**, i nie dotyka obiektu **tor3** – postarajmy się ułożyć to zadanie z bloków.

1 Blok **call tor1.CollidingWith other** zwraca prawdę, gdy tor1 koliduje z obiektem, który doczepimy do pola **other** w tym bloku. Doczepić powinniśmy tam **Samochod**.



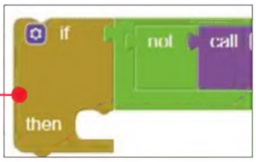
2 Ponieważ nas interesuje moment, gdy kolizji między obiektami już nie ma – z przodu doczepiamy zaprzeczenie – czyli blok **not** z kategorii **Logic**.

3 Analogiczny skrypt układamy dla dwóch pozostałych fragmentów toru.



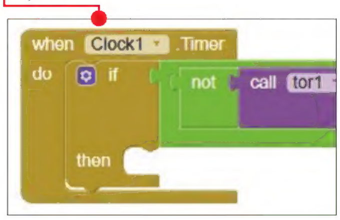
4 Z kategorii **Logic** bierzemy też dwa spójniki and – wkładamy jeden w drugi – w ten sposób dostaniemy blok, który pozwoli na połączenie trzech warunków – które muszą być jednocześnie spełnione, by coś się stało.

5 Wkładamy trzy bloki utworzone w kroku **3** do slotów w konstrukcji z kroku **4**.



6 Całą konstrukcję dołączamy do bloku **if** z kategorii **kontrol**. If to instrukcja warunkowa – to co włożymy do jej środka, wykona się, gdy warunek przypięty u góry będzie spełniony.

7 If wkładamy w blok **when Clock1.Timer do**, by warunek był sprawdzany z każdym wywołaniem zdarzenia Timera, czyli co 0,01 s.



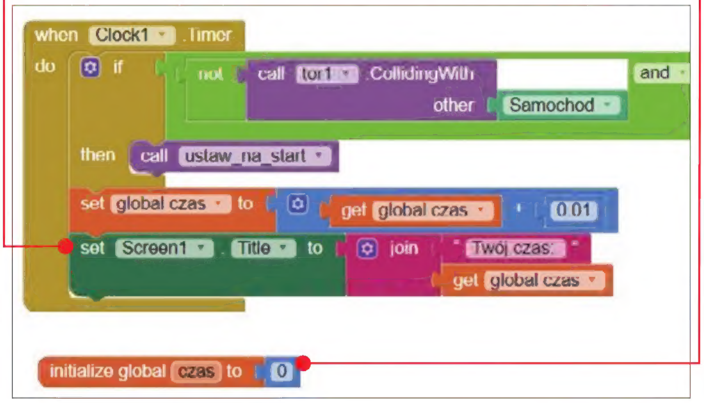
8 Gdy warunek z if będzie spełniony, powinna wykonać się procedura **ustaw_na_start**, by samochód wrócił na pozycję startową.

ODLICZANIE CZASU GRY

O jakości przeprowadzonej rozgrywki powinien decydować czas dojechania do mety.

1 Tworzymy zmienną o nazwie **czas** (w zmiennych możemy przechowywać różne wartości). W tym celu bierzemy pomarańczowy blok ● z kategorii **Variables** i wykorzystujemy go do stworzenia zmiennej **czas** ● o wartości początkowej **0**.

2 Z każdym przejściem pętli należy zwiększać wartość zmiennej **czas** o **0,01** i wyświetlać ją na pasku tytułowym, który widoczny jest nad **Canvas** podczas gry. Zrobimy to z wykorzystaniem bloku **set Screen1.Title to** ●.

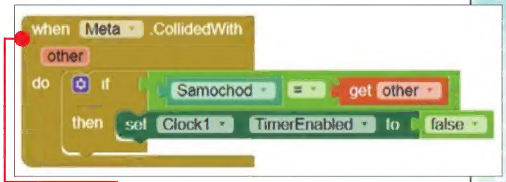


Sugestie

Dobrym sposobem na doskonalenie swoich umiejętności jest praca samodzielna – już bez instrukcji. Spróbujmy rozbudować grę o własne pomysły. Jeśli nam ich brakuje, oto kilka sugestii:

- Dodajmy przycisk do restartu gry, który pojawi się, gdy samochód dojedzie do mety i zniknie po kliknięciu, pozwalając na przeprowadzenie kolejnej rundy.
- Dodajmy nowe fragmenty toru – wprowadzając więcej zakrętów, które utrudnią szybkie przejechanie całej trasy.
- Dodajmy drugi poziom gry, który będzie pojawiał się, gdy samochód dojedzie do mety.

ZATRZYMANIE CZASU NA MECIE

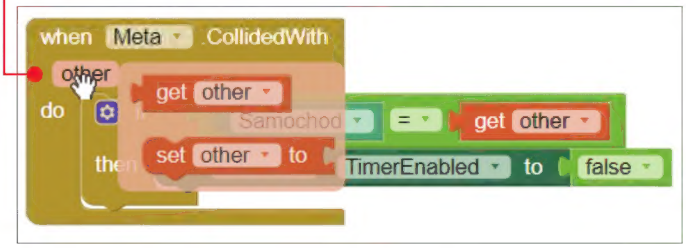


Na koniec powinniśmy zaprogramować to, co ma się dziać, gdy samochód dojedzie do mety.

1 Z kategorii **Meta** bierzemy blok **when Meta.CollidedWith other do** ●, i wkładamy do niego nowy if.

2 W if przypinamy warunek, że samochód jest tym obiektem, z którym koliduje **Meta**. Informacja o obiekcie jest ukryta w bloku **other**, wyświetlanym, gdy najedziemy kursorem na pole **other** ●.

3 Gdy warunek jest spełniony, należy wyłączyć wykonywanie pętli przez komponent **Clock1**. Pamiętajmy o tym, by na bieżąco testować działanie projektu. Efekt powinien wyglądać tak ●.

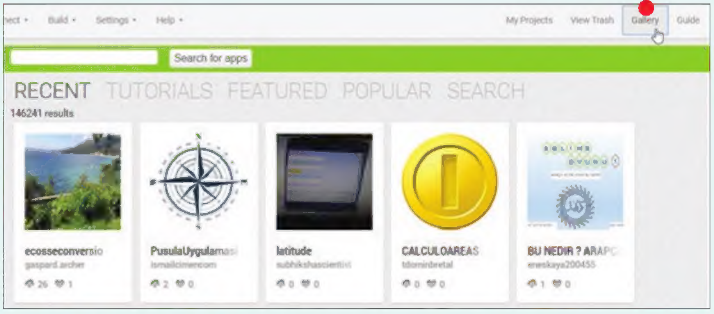


Galeria projektów

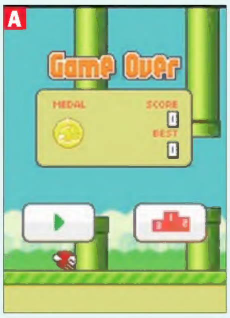
App Inventor pozwala na publikację swoich projektów. Dzięki temu powstaje galeria. W niej także można poszukać inspiracji do własnych gier. Aby wejść do galerii, należy kliknąć na przycisk **Gallery** ● na pasku menu głównego. W galerii możemy znaleźć między innymi:

- klony popularnych gier – na przykład **Flappy Bird** **A**;
- gry typu kliker – na przykład **Pizza Clicker** **B**,
- ale też programy – jak stoper, symulator kostki do gry czy kalkulator **C**.

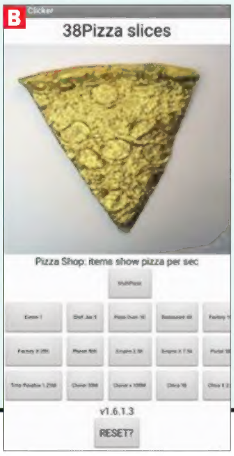
To tylko kilka przykładów projektów, jakie można tam znaleźć. Nasze projekty również mogą trafić do galerii. Rozwijamy opcję **My projects** z menu głównego i wybieramy pozycję **My projects** ●. W ten sposób dostaniemy się do zestawiania wszystkich projektów zrealizowanych na koncie, poprzez które logujemy się do programu. Zaznaczamy projekt, który chcemy udostępnić, i klikamy na przycisk **Publish to Gallery** ●. W kolejnym oknie uzupełniamy informacje o projekcie i zatwierdzamy publikację przyciskiem **PUBLISH** ●.



A



B



C